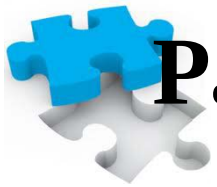

PatternApplier

Manual de Usuario

Índice:

Introducción **4**

Conceptos fundamentales **4**

1. Instalación de la herramienta **5**

1.1 Descargar desde el repositorio svn ASLab 5

1.1 Importar RAS Asset 7

2. Aplicación de patrones a modelos **9**

2.1 Apertura del modelo y los patrones 9

2.2 Enlazado del modelo con los patrones 9

2.3 Ejecución del pluglet 12

3. Diseño de patrones **15**

3.1 Parámetros de plantilla 15

3.1.1 Adición de parámetros a nuestro patrón 15

3.1.2 Limitaciones de parámetros de plantilla 17

3.2 Perfil para especificación de transformaciones 17

3.2.1 Uso de estereotipos 17

3.2.2 Estereotipos disponibles 20

3.2.3 Limitaciones de los estereotipos 20

3.3 Renombrar elementos mediante parámetros 21

4. Ampliación de la funcionalidad **23**

4.1 Extender elementos parametrizables 23

4.2 Extender elementos estereotipados 25

4.3 Clase UMLTools 26

4.4 Clase CheckTools 27

Índice de ilustraciones:

Ilustración 1. Descarga de PatternApplier desde repositorio 5
Ilustración 2. Check out (paso 1) 6
Ilustración 3. Check out (paso 2) 6
Ilustración 4. Importar 7
Ilustración 5. Importar RAS Asset 7
Ilustración 6. Seleccionar localización de PatternApplier.ras 8
Ilustración 7. Finalizar importación de RAS Asset 8
Ilustración 8. Modelos abiertos y cerrados 9
Ilustración 9. Añadir Template Binding 10
Ilustración 10. Seleccionar patrón con el que enlaza el Template Binding 10
Ilustración 11. Añadir Actual Parameter (substituciones) 11
Ilustración 12. Seleccionar Actual Parameter para una plantilla 11
Ilustración 13. Template Binding con todas las substituciones realizadas 12
Ilustración 14. Ejecutar PatternApplier como Pluglet por primera vez 12
Ilustración 15. Ejecutar PatternApplier 13
Ilustración 16. Consola y explorador de paquetes tras ejecución de PatternApplier 13
Ilustración 17. Añadir Template Parameter 15
Ilustración 18. Vista de propiedades de un TemplateSignature 16
Ilustración 19. Vista de propiedades de un Template Parameter 16
Ilustración 20. Vista de una clase parametrizada frente a otra clase sin parametrizar 16
Ilustración 21. Añadir perfil a un modelo 17
Ilustración 22. Seleccionar origen del perfil a añadir 18
Ilustración 23. Seleccionar el perfil a añadir 18
Ilustración 24. Aplicar estereotipos 19
Ilustración 25. Seleccionar estereotipos a aplicar 19
Ilustración 26. Ejemplo de elementos con estereotipos del Perfil PatternSpec aplicados 19
Ilustración 27. Ejemplo de patrón con renombre de elementos a través de parámetros 21

Ilustración 28. Resultado de aplicar patrón con renombre de elementos a través de parámetros 22

Ilustración 29. Ejemplo de extensión de parámetros de plantilla y elementos estereotipables 23

Ilustración 30. Diagrama de funcionamiento de la clase *PatternParameters* 24

Ilustración 31. Diagrama de creación de elementos nuevos en el modelo de destino 25

Índice de códigos:

Código 1. Extensión de tipos de parámentros de plantilla 24

Código 2. Forma del método solveOperation 24

Código 3. Extensión de tipos de elementos que se pueden crear en el modelo de destino 26

Código 4. Forma del método copyEnumeration en la clase UMLTools 26

Código 5. Forma del método searchEnumeration en la clase CheckTools 28

Introducción

PatternApplier es una herramienta de diseño que permite transformar un modelo UML aplicando uno o varios patrones sobre él. La herramienta está desarrollada como un pluglet para ejecutar en el entorno de modelado de Rational Software Architect de IBM, y permite usar patrones especificados mediante modelos UML en este entorno.

El presente manual muestra al usuario de la herramienta *PatternApplier* cómo usarla en sus modelos utilizando patrones ya definidos, así como la manera de diseñar sus propios patrones o de ampliar la funcionalidad de la herramienta. Se supone que el usuario está familiarizado con RSA y el modelado con UML.

Conceptos fundamentales

- **Modelo:** Es una abstracción de un sistema. El modelo describe los aspectos del sistema que son relevantes para el propósito del modelo, a un nivel de detalle apropiado. En este documento nos referiremos con el término "modelo" a modelos de sistemas técnicos capturados mediante el lenguaje de modelado UML.

- **Proyecto:** Los proyectos a los que se hace referencia en el documento son proyectos de modelado de RSA. Un proyecto de modelado en RSA es un proyecto en el espacio de trabajo de RSA que contiene uno o varios modelos UML y, opcionalmente, uno o varios perfiles UML, así como cualquier otro tipo de recursos auxiliares: ficheros con imágenes, de texto, etc.

- **Patrón:** Es una solución a un problema de diseño. Los patrones mencionados en el documento son soluciones de diseño que capturamos en modelos UML. Así pues, tendremos un modelo de definición del patrón.

- **Rol:** Es un papel o función que cumple algo o alguien en un contexto. En la definición de un patrón, varios elementos (parámetros) realizan distintos roles en el contexto del patrón para resolver el problema. Al aplicar un patrón a un modelo, se especifican qué elementos del patrón dan valor a sus parámetros, es decir, realizan los roles definidos en el patrón.

- **TemplateParameter:** Es un elemento UML que permite definir una plantilla con parámetros (véase [UML2-2009] pág. 628). Se usarán en los modelos de definición de los patrones para definir roles que tendrán que ser jugados por elementos externos al patrón.

- **TemplateBinding:** Es un elemento UML que permite enlazar un modelo con otro modelo que contenga plantillas con parámetros (véase [UML2-2009] pág. 627). A través de este enlace se puede saber elementos que juegan los roles.

1. Instalación de la herramienta

A continuación se explican dos maneras de importar la herramienta *PatternApplier* al explorador de proyectos del RSA para poder trabajar con ella. Podremos descargar las fuentes del repositorio svn de ASLab o importarla desde un RAS Asset.

1.1 Descargar desde el repositorio svn ASLab

En la vista **SVN Repository Exploring**, buscamos el directorio **/root/people/ealarcon/PatternApplier**[1] en el repositorio SVN de ASLab. Pinchamos con el botón secundario del ratón y seleccionamos **Checkout.**

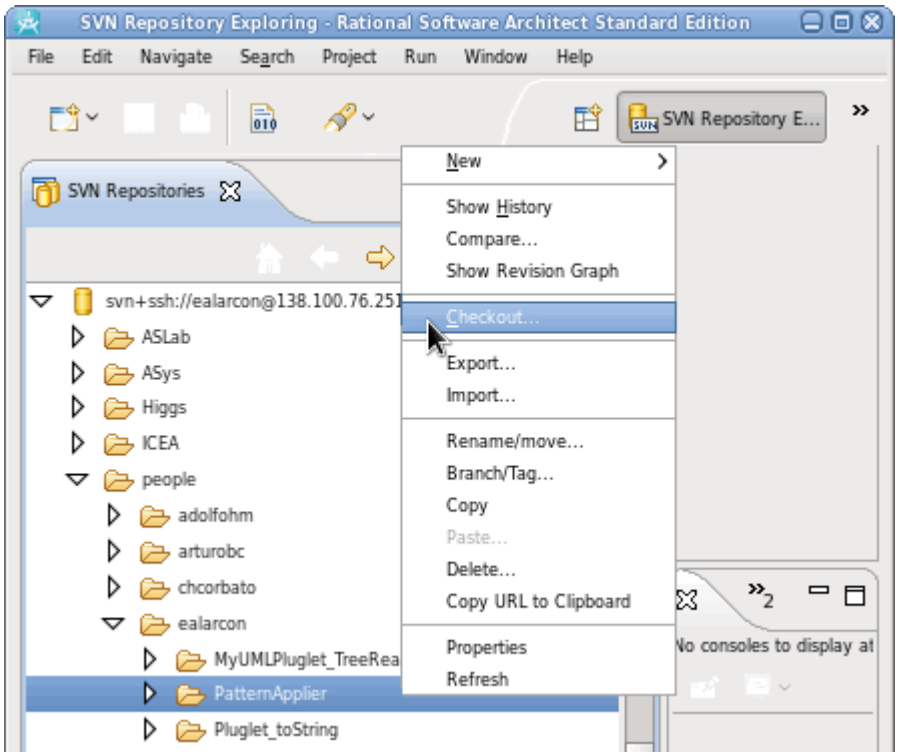

Ilustración 1. Descarga de PatternApplier desde repositorio

En las siguientes ventanas elegimos las opciones que sean de nuestro interés. Será necesario marcar que la salida ha de ser un proyecto en el espacio de trabajo. ***(Check out as a Project in the workspace).***

Al final se realiza la descarga del proyecto al explorador de proyectos.

[1] URL del repositorio: svn+ssh://software.aslab.upm.es/home/svnroot

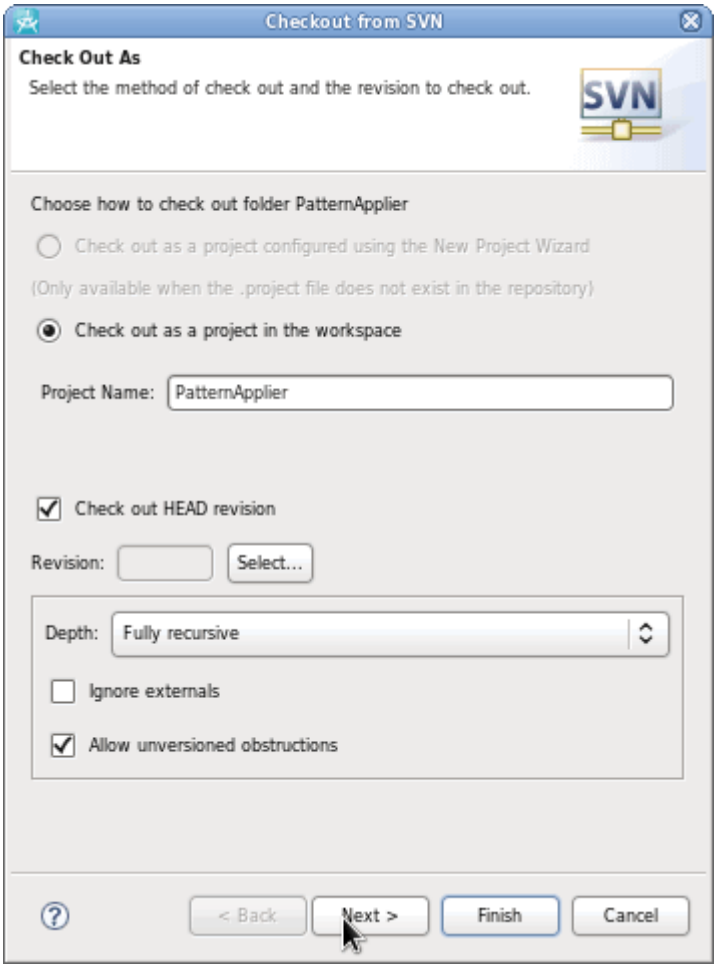
Checkout from SVN
Check Out As
Select the method of check out and the revision to check out.
SVN
Choose how to check out folder PatternApplier
Check out as a project configured using the New Project Wizard
(Only available when the .project file does not exist in the repository)
Check out as a project in the workspace
Project Name: PatternApplier
Check out HEAD revision
Revision:
Select...
Depth: Fully recursive
Ignore externals
Allow unversioned obstructions
< Back
Next >
Finish
Cancel

Ilustración 2. Check out (paso 1)

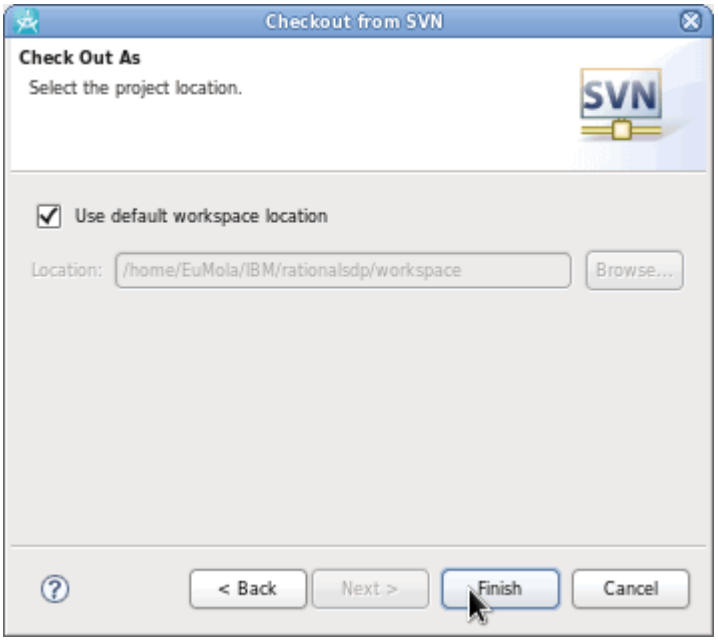

Ilustración 3. Check out (paso 2)

1.1 Importar RAS Asset

Otra opción es importar el RAS Asset en el que está empaquetada la herramienta. En primer lugar debemos descargar en nuestro equipo el fichero que lo contiene de http://aslab.org/software/PatternApplier.ras. Seguidamente tenemos que importar el contenido en nuestro RSA. Para ello, seleccionamos **File/Import…**

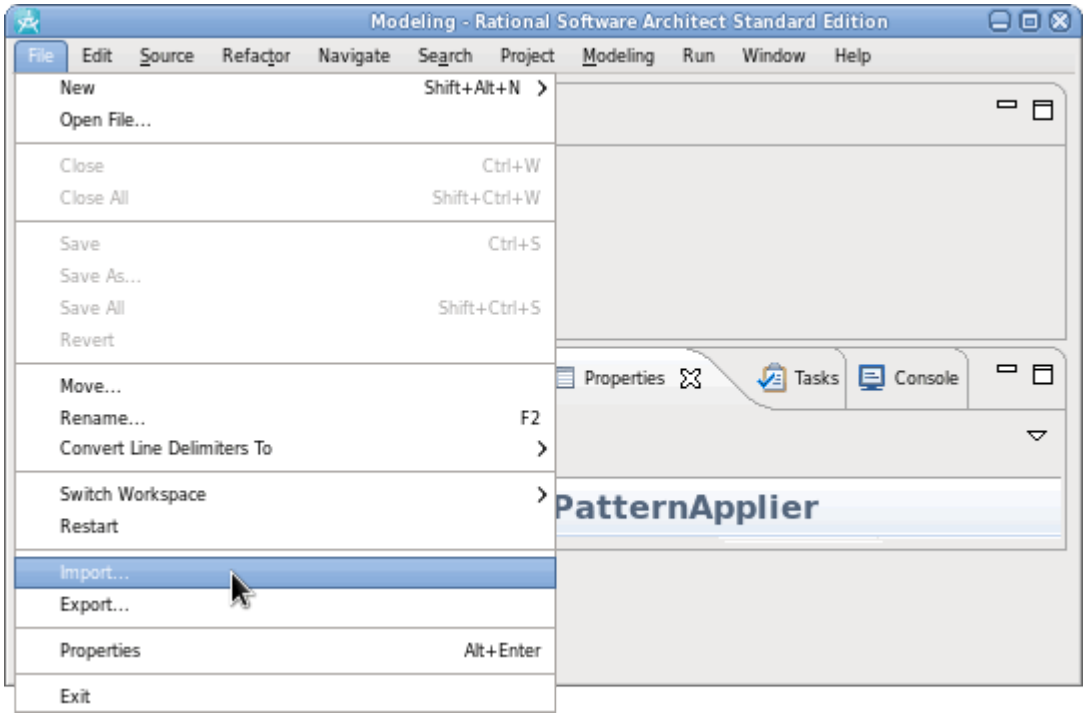

Ilustración 4. Importar

En la ventana que surge, elegimos **RAS Import** y continuamos.

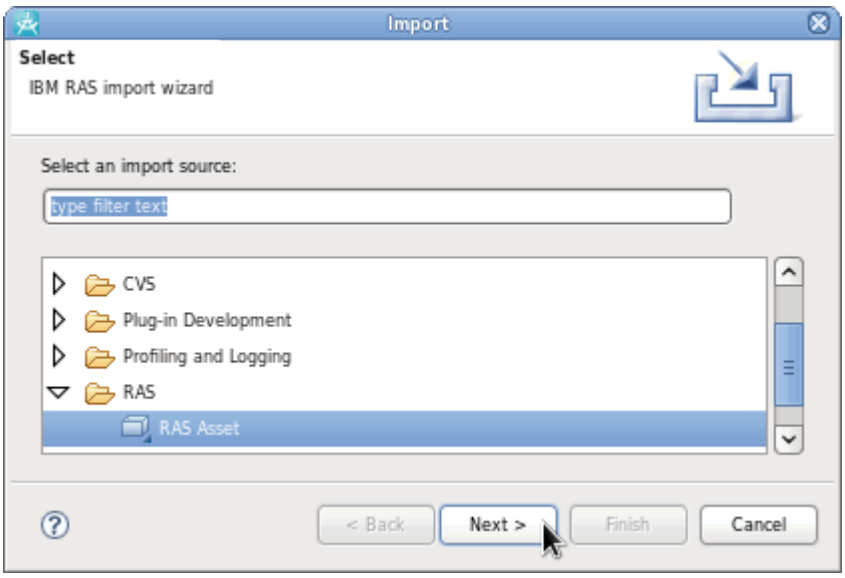

Ilustración 5. Importar RAS Asset

A continuación indicamos la localización del fichero que contiene el RAS Asset, y finalmente aceptamos el destino pinchando **finish**.

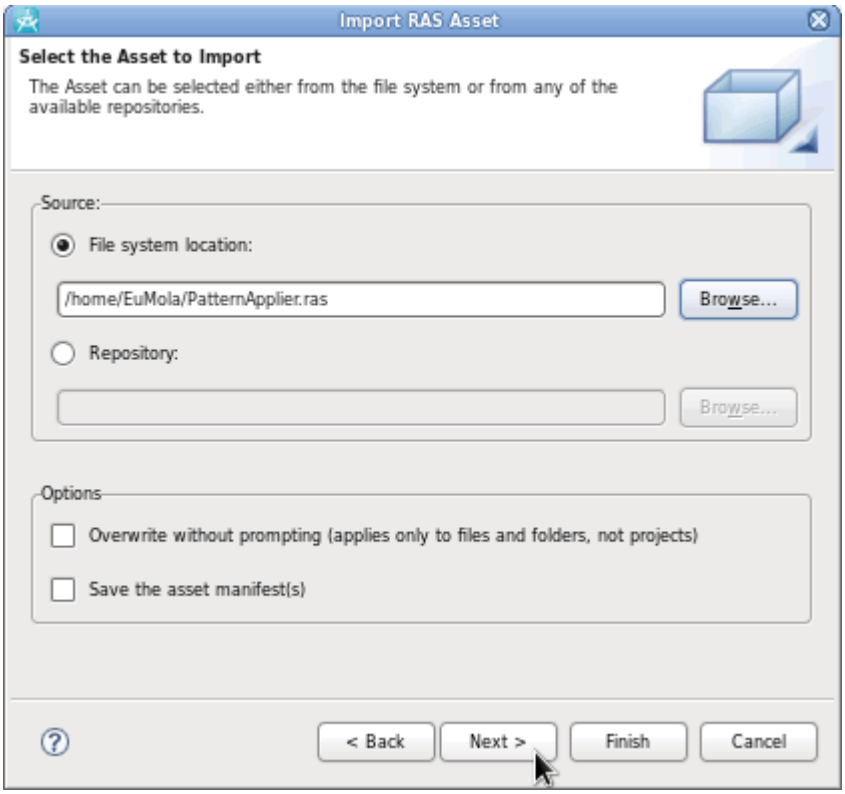

Ilustración 6. Seleccionar localización de PatternApplier.ras

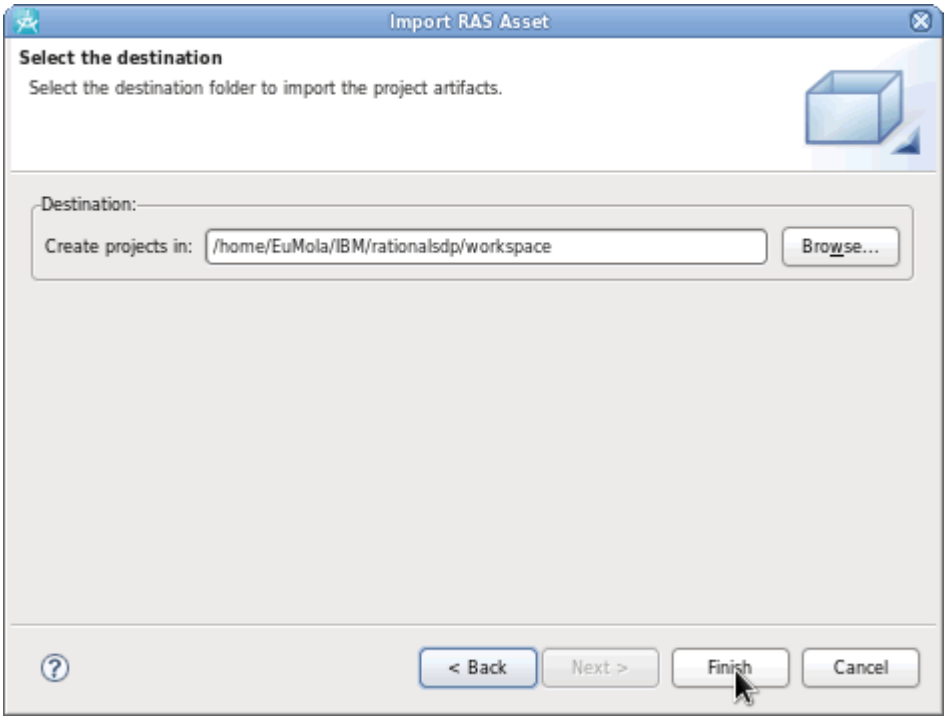

Ilustración 7. Finalizar importación de RAS Asset

2. Aplicación de patrones a modelos

Al usar la herramienta sobre un modelo, se generará automáticamente un nuevo modelo (o transformación del original, dependiendo de la opción elegida) resultante de la aplicación del patrón o patrones sobre éste. A continuación se explican los pasos necesarios para ejecutar el pluglet.

2.1 Apertura del modelo y los patrones

Para aplicar los patrones al modelo es necesario haber abierto tanto el modelo como los patrones (que son modelos UML en RSA también).

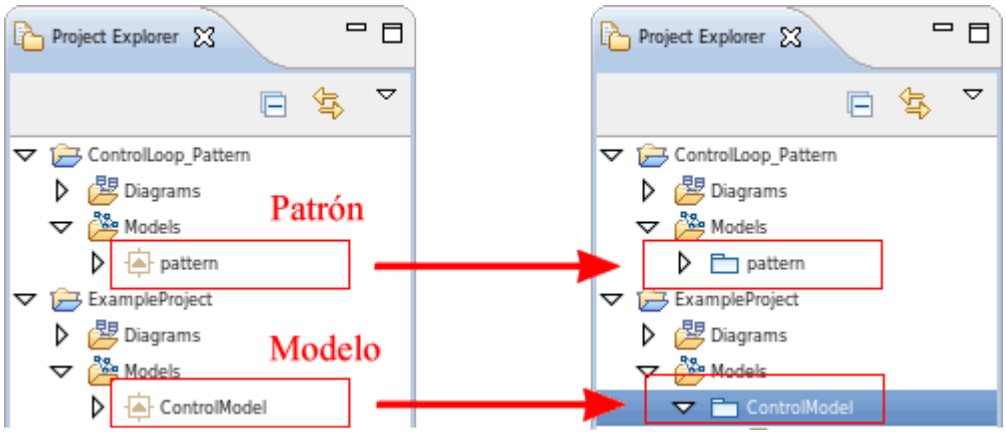

Ilustración 8. Modelos abiertos y cerrados

A la izquierda de la figura se muestran dos proyectos de modelado del espacio de trabajo de RSA: *ControlLoop_Pattern*, que contiene un modelo denominado *pattern*, y *ExampleProject*, que contiene un modelo denominado *ControlModel*.

El modelo del primero es la definición del patrón que se quiere aplicar en el modelo *ControlModel*. A la derecha de la figura se muestra el resultado de abrir dichos modelos (siempre en la perspectiva de modelado de RSA).

2.2 Enlazado del modelo con los patrones

Para enlazar el modelo con un patrón añadimos al modelo un *Template Binding* por cada patrón a aplicar sobre éste. Para ello, en el explorador de proyectos, pinchamos el modelo con el botón secundario y seleccionamos **Add UML/Template Binding.**

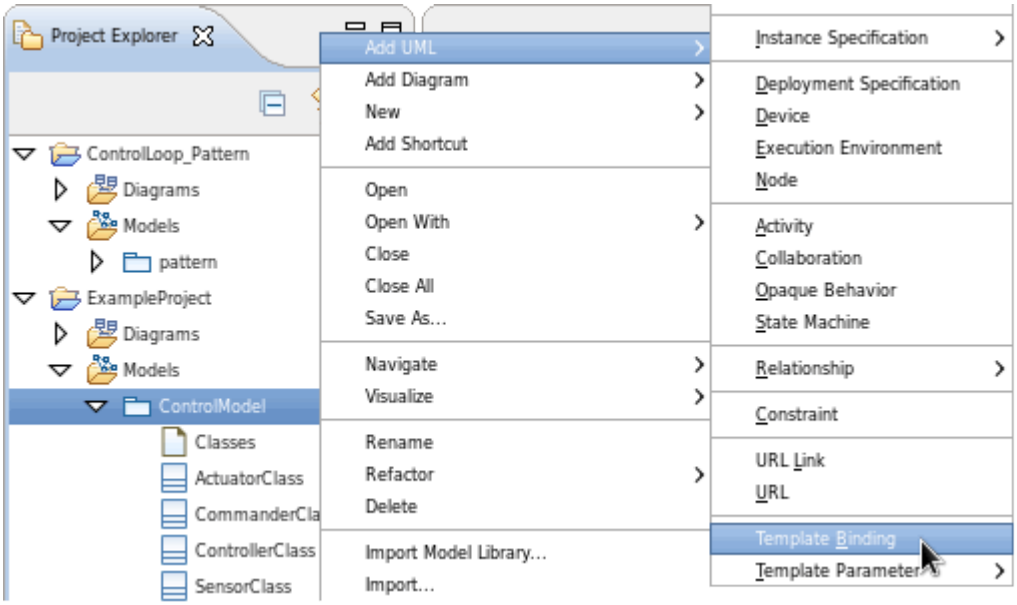

Ilustración 9. Añadir Template Binding

Aparecerá una ventana en la que buscaremos el patrón que queremos aplicar.

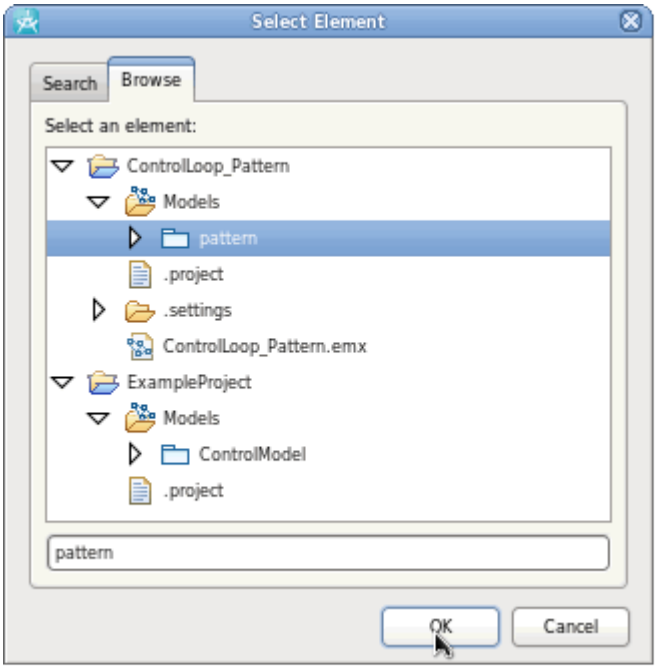

Ilustración 10. Seleccionar patrón con el que enlaza el Template Binding

En las propiedades del *Template Binding* que se ha creado aparecerán los parámetros de la plantilla del patrón (roles). Para ello, pinchamos el elemento *TemplateBinding* que hemos añadido *(Pattern)* y seleccionamos la pestaña de la vista de propiedades. Seleccionamos una de los parámetros ***(controller)*** y pinchamos **Browse.**

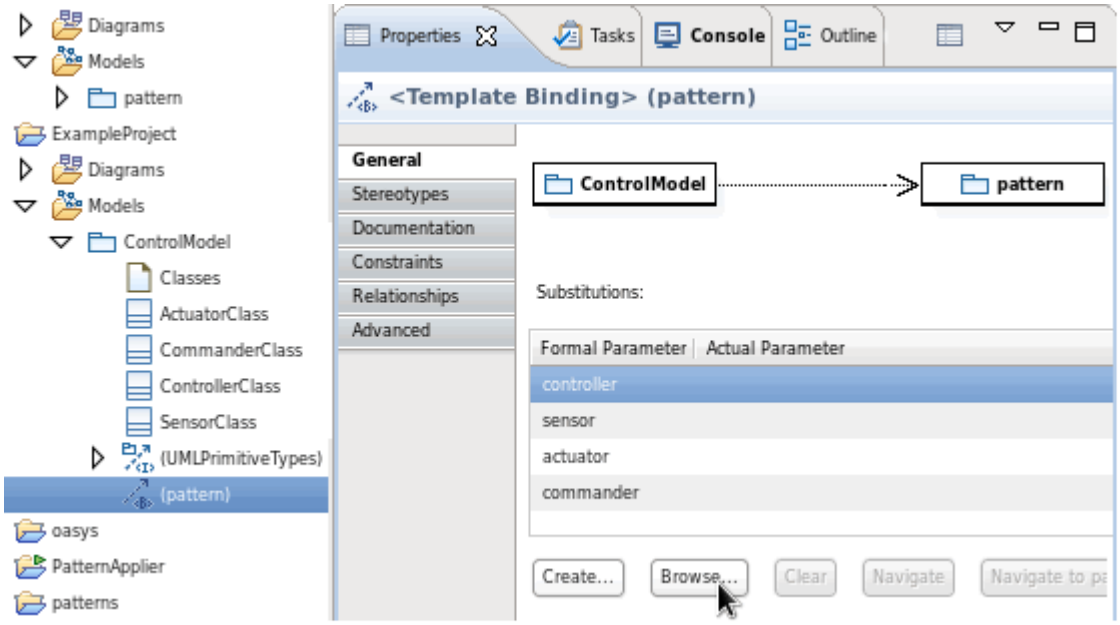

Ilustración 11. Añadir Actual Parameter (substituciones)

Seleccionaremos por cada parámetro tantos elementos del modelo queramos que asuman dicho rol del patrón pudiendo no seleccionar ninguno en el caso de que el patrón contemple dicha posibilidad. Cuando más de un elemento pueda asumir un rol, en la misma ventana, haremos una selección múltiple de los elementos (manteniendo pulsado **Ctrl** mientras seleccionamos los elementos del modelo).

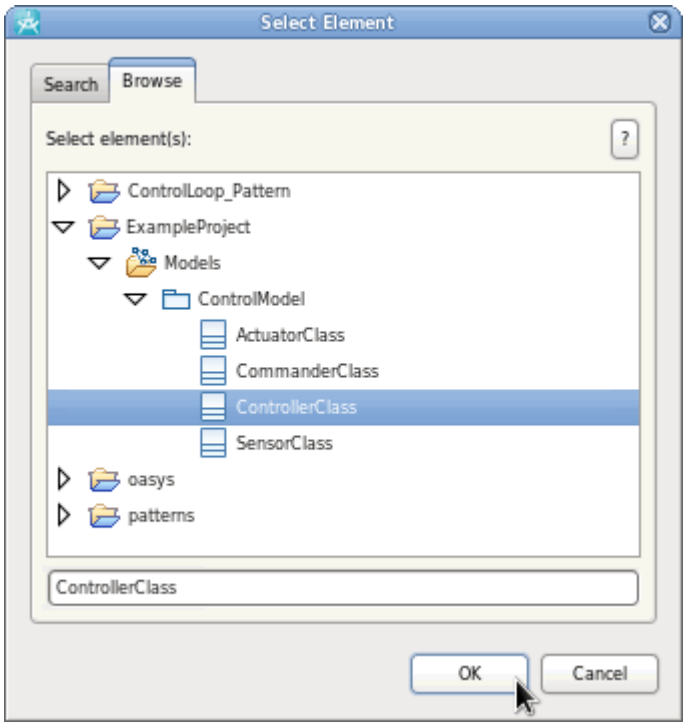

Ilustración 12. Seleccionar Actual Parameter para una plantilla

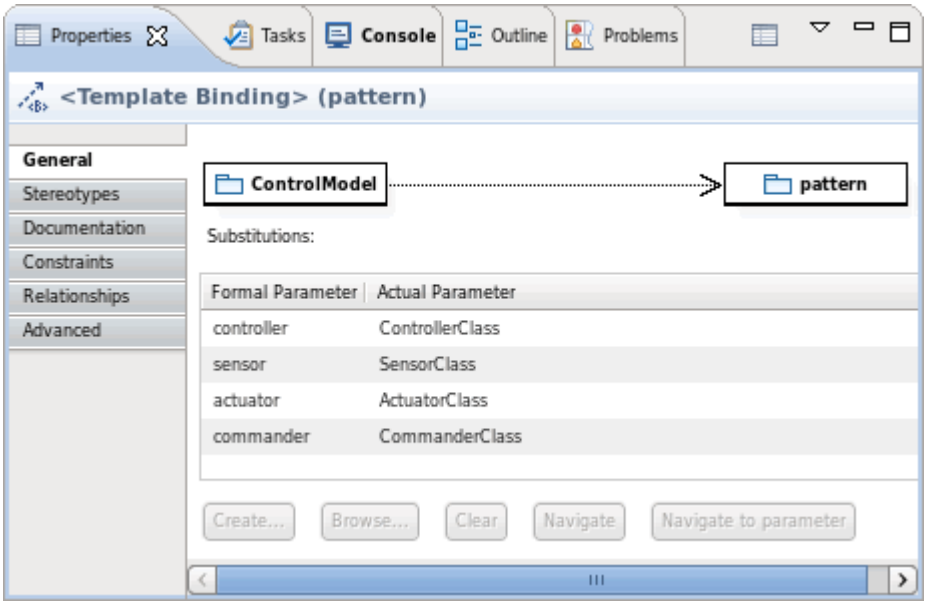

Ilustración 13. Template Binding con todas las substituciones realizadas

2.3 Ejecución del pluglet

Una vez creados todos los enlaces a los distintos patrones y asignados todos los parámetros reales podemos proceder a ejecutar la herramienta. Si es la primera vez que se ejecuta *PatternApplier*, la información relativa a la configuración de su ejecución no estará registrada en nuestro RSA, por lo que habrá que realizar una primera ejecución de la siguiente manera: en el explorador de proyectos desplegamos el proyecto **PatternApplier**, y en el paquete **org.aslab.asys.models.patterns.tools** pinchamos con el secundario sobre **PatternApplier.java** y seleccionamos **Run As/Pluglet**[2]**.**

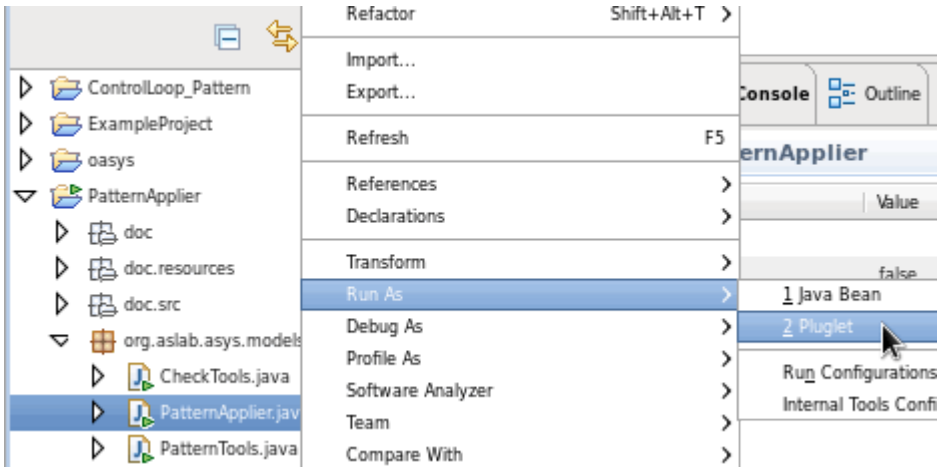

Ilustración 14. Ejecutar PatternApplier como Pluglet por primera vez

[2] Si esta opción no aparece puede que debas habilitar las capacidades de desarrollo de plugins en las preferencias del RSA: Help->Preferences->Capabilities

La configuración de ejecución quedará registrada y la aplicación podrá ser lanzada de manera que se identifique como parámetro de entrada del pluglet el modelo sobre el que se aplica el patrón. Para ello, en el explorador de proyectos, seleccionamos el modelo. A continuación (sin marcar ningún otro elemento en cualquier parte del entorno del RSA) seleccionamos en la barra principal **Run/Internal Tools/PatternApplier.**

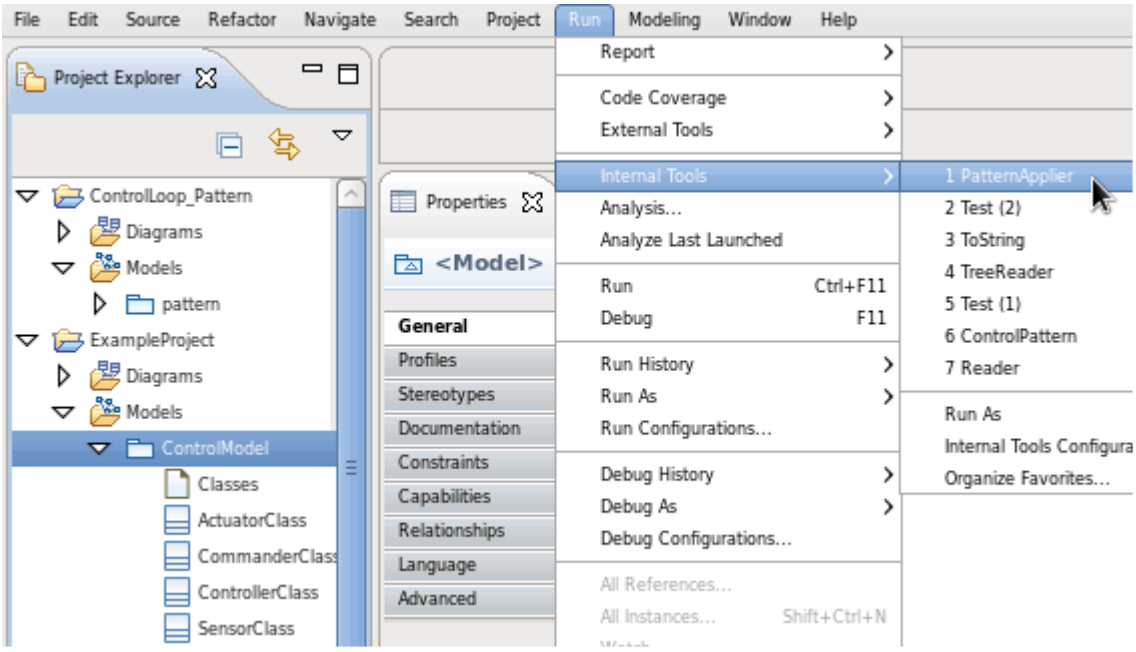

Ilustración 15. Ejecutar PatternApplier

Si el modelo seleccionado es válido aparecerá una ventana para introducir el nombre del nuevo modelo generado (en caso de no querer alterar el original). Finalmente, las transformaciones sobre el modelo se realizarán. Podremos comprobar en la vista de consola si el proceso ha sido correcto.

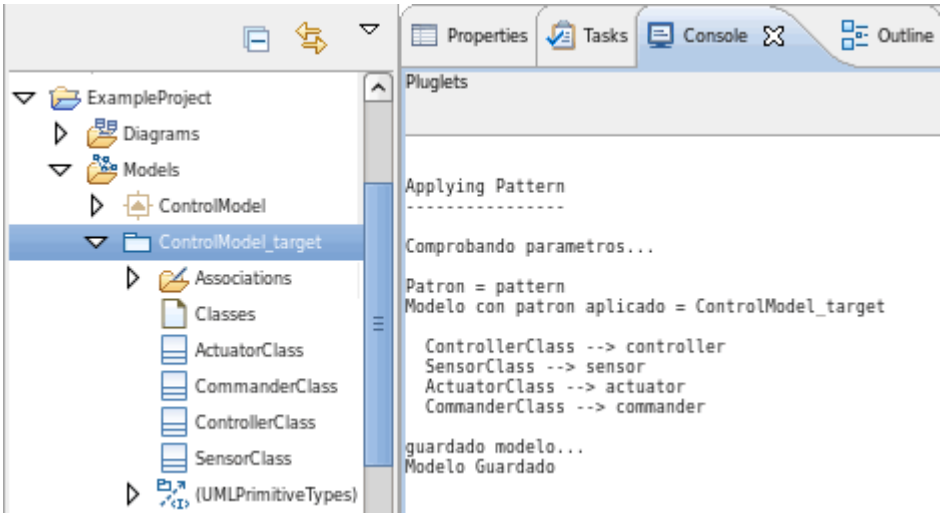

Ilustración 16. Consola y explorador de paquetes tras ejecución de PatternApplier

En el caso de haber elegido un modelo de destino nuevo, en el explorador de proyectos aparecerá un nuevo modelo *(ControlModel_target)* en el proyecto de modelado *(ExampleProyect)* que contenía el modelo de partida *(ControlModel)*. Este nuevo modelo será el resultado de aplicar el patrón al modelo inicial. Si no se eligió un modelo de destino nuevo, el patrón se aplicará sobre el modelo inicial *(ControlModel)*.

3. Diseño de patrones

La herramienta transforma modelos UML de acuerdo con la especificación del patrón, que se realiza siguiendo un lenguaje propio de especificación de patrones que hace uso del UML2 y un perfil del mismo, el *PatternSpec.*

Así, los dos elementos fundamentales del lenguaje de especificación de patrones son dos elementos de UML: los parámetros de plantilla *(template parameters)* y los estereotipos del perfil *PatternSpec*. Estos completan el diseño de los patrones a la par que los dotan de funcionalidad en su interacción con la herramienta.

3.1 Parámetros de plantilla

Al incluir un parámetro en el diseño del patrón estaremos modelando un elemento que posteriormente será substituido por otro, es decir, uno o varios elementos del modelo asumirán el rol del elemento que modela el parámetro. La herramienta sigue así parcialmente la recomendación de la OMG para la especificación de patrones mediante colaboraciones parametrizadas (véase [Booch-2005] Capítulo 29). Con la inclusión de los parámetros necesarios tendremos en el patrón una plantilla con parámetros que se deben asignar a la hora de aplicar el patrón sobre un modelo.

3.1.1 Adición de parámetros a nuestro patrón

Para añadir un parámetro al patrón, pichamos con el secundario el modelo de definición del patrón y seleccionamos **Add UML/Template Parameter/(tipo elemento)**[3]**.**

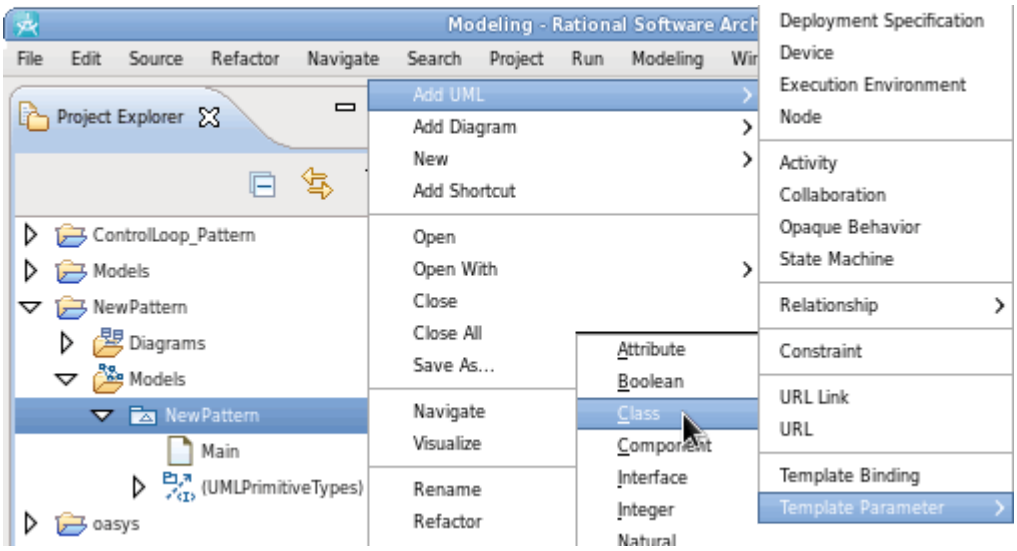

Ilustración 17. Añadir Template Parameter

[3] Ver apartado 3.1.2 Limitaciones

En el modelo de definición del patrón aparecerá un *TemplateSignature* (firma de platillas, visible en el explorador de proyectos. Dentro, aparecerán distintos tipos de plantillas dependiendo del tipo de parámetro que se haya seleccionado.

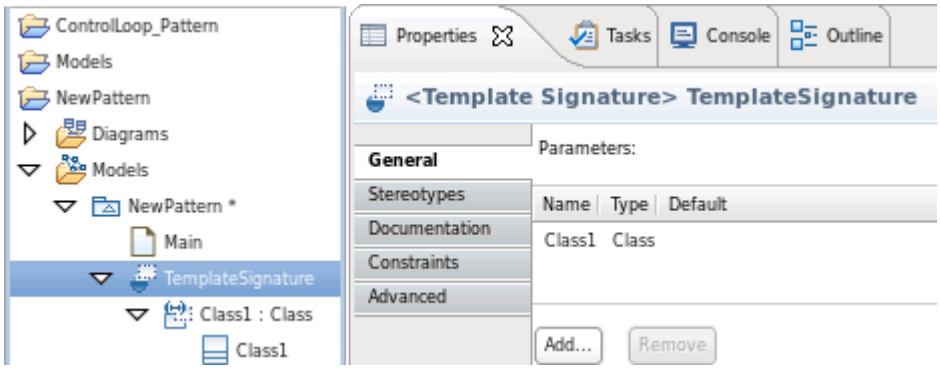

Ilustración 18. Vista de propiedades de un TemplateSignature

Cada plantilla, contiene un elemento (elemento parametrizado) del tipo del *template parameter* que podrá ser modelado en base al rol que representa dentro del patrón. Este rol será asumido por otros elementos al aplicar el patrón sobre un modelo.

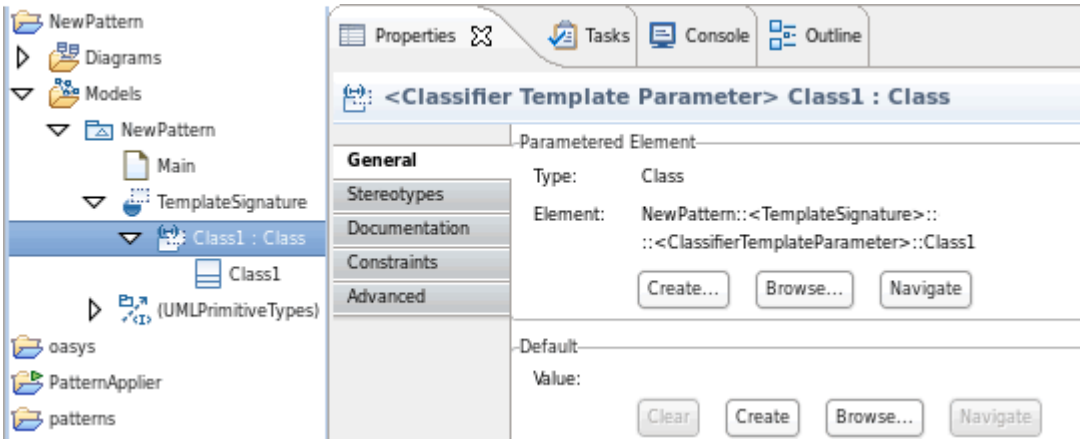

Ilustración 19. Vista de propiedades de un Template Parameter

En el diagrama de clases, tienen la misma apariencia un elemento sin parametrizar (Class2) que uno parametrizado del mismo tipo (Class1).

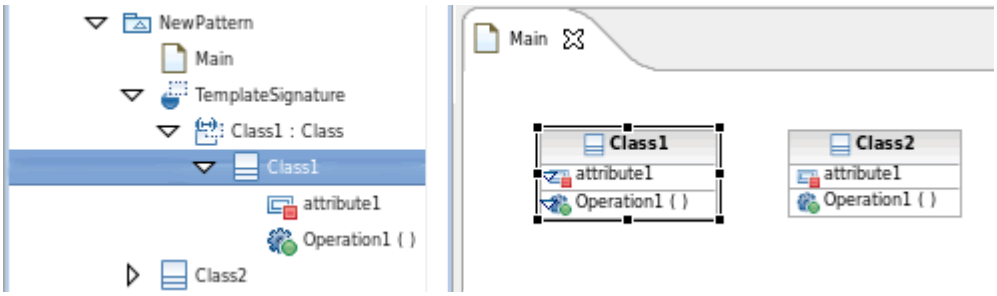

Ilustración 20. Vista de una clase parametrizada frente a otra clase sin parametrizar

3.1.2 Limitaciones de parámetros de plantilla

El uso de parámetros en patrones para ser aplicados con la herramienta *PatternApplier* está limitado a elementos del tipo ***Class*** (clase) e ***Interface*** (interfaz). Los elementos como atributos y operaciones, pertenecientes a dichos elementos deberán ser tratados usando el mecanismo explicado en el siguiente punto. De manera que habrá que especificar si son creados o no en el elemento que juega el rol.

Si se quiere poder utilizar otro tipo de elementos como parámetros será necesario ampliar la funcionalidad de la herramienta.

3.2 Perfil para especificación de transformaciones

El uso de perfiles UML permite la extensión de la sintaxis y la semántica del UML para expresar conceptos específicos de un determinado domino de aplicación. En nuestro caso de diseño de patrones, definir una notación para extender UML como lenguaje formal de especificación de patrones. Es "formal" porque especifica las transformaciones que realizará la herramienta *PatternApplier*.

El perfil *PatternSpec*, contenido en la carpeta de proyecto de la herramienta, ha sido definido para proveer de estas notaciones necesarias para los patrones. Uno de los mecanismos usados en la definición de perfiles son los estereotipos. Estos son unos tipos limitados de metaclases que se usan en combinación con otro tipo de metaclases a las que extienden.

3.2.1 Uso de estereotipos

En el perfil *PatternSpec* hay definidos diferentes estereotipos. En el modelo de definición del patrón, se aplicará un estereotipo a un elemento en función de lo que se quiera que la herramienta haga con ese elemento (por ejemplo, crearlo en el modelo final).

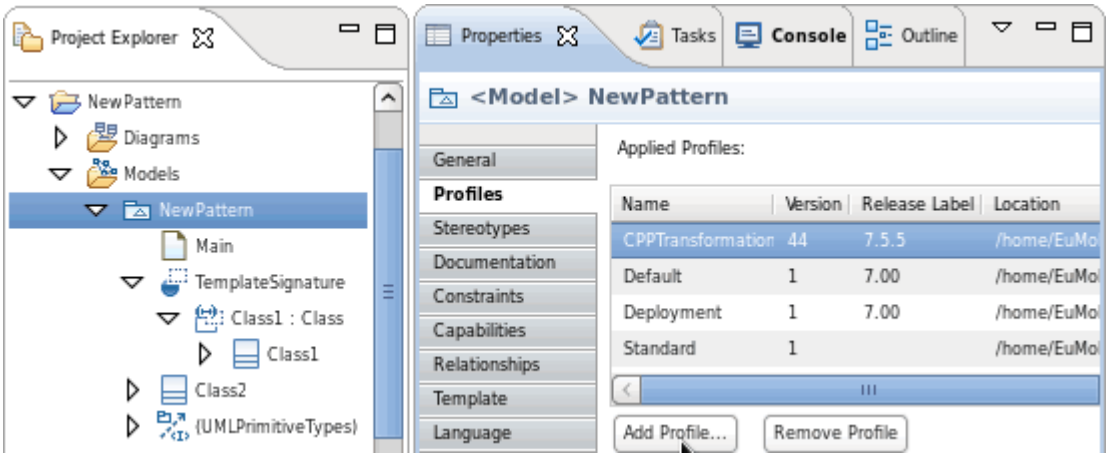

Ilustración 21. Añadir perfil a un modelo

Para poder estereotipar los elementos del patrón es necesario añadir el perfil al modelo de definición del patrón (Ilustración 21). Marcamos el **modelo en el explorador de proyectos**; en la pestaña de **propiedades** seleccionamos **perfiles** y pulsamos **Add Profile…**

Seleccionamos la opción de **perfil en el espacio de trabajo.**

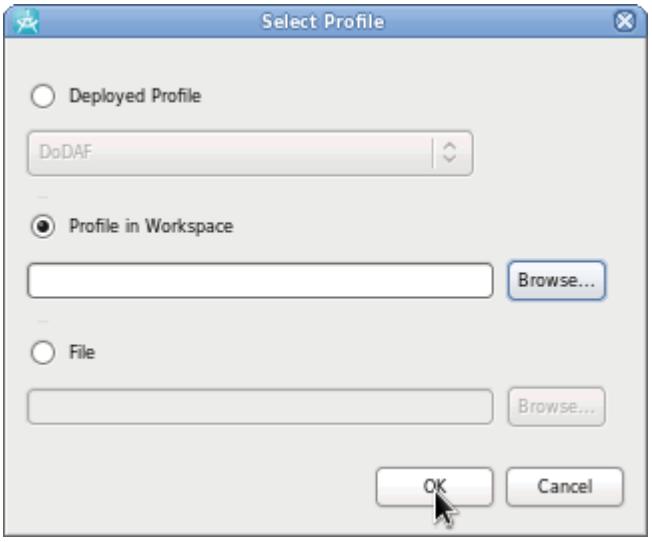

Ilustración 22. Seleccionar origen del perfil a añadir

Elegimos el perfil ***PatternSpec*** en la **carpeta *PatternApplier***.

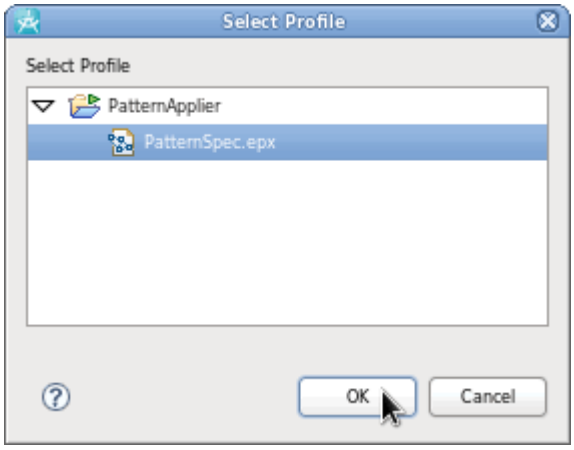

Ilustración 23. Seleccionar el perfil a añadir

Ahora ya podemos estereotipar los elementos que definen el patrón. Para ello, marcamos el **elemento** y en la pestaña de **propiedades**, seleccionamos **estereotipos** y pulsamos **Apply stereotypes…**

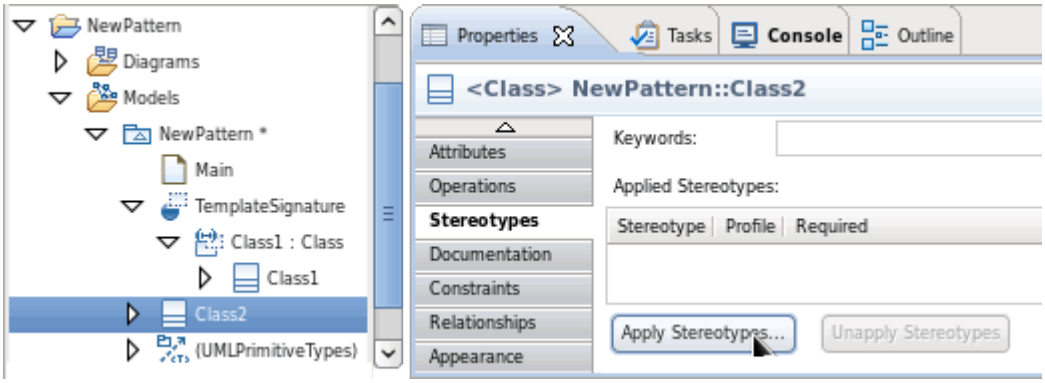

Ilustración 24. Aplicar estereotipos

Aparecerá una ventana con un listado de los estereotipos que se pueden aplicar a ese tipo de elemento. Al lado del nombre del estereotipo aparece el nombre del perfil, por lo que será fácil identificar los estereotipos aportados por el perfil *PatternSpec*.

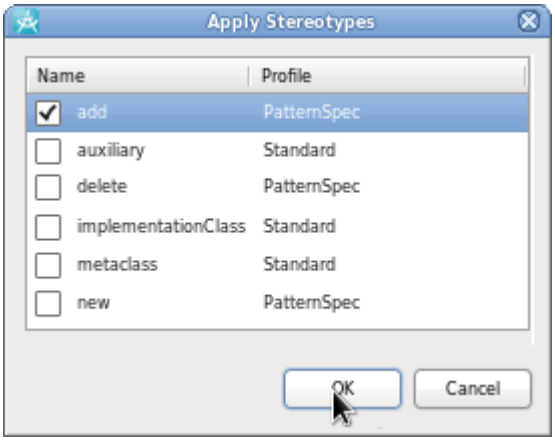

Ilustración 25. Seleccionar estereotipos a aplicar

Tras seleccionar un estereotipo, el elemento se mostrará estereotipado tanto en el explorador de proyectos como en el diagrama. En la siguiente figura se muestran además de una clase parametrizada (*Class1*), una clase y una asociación estereotipadas *(Class2 y Association.*

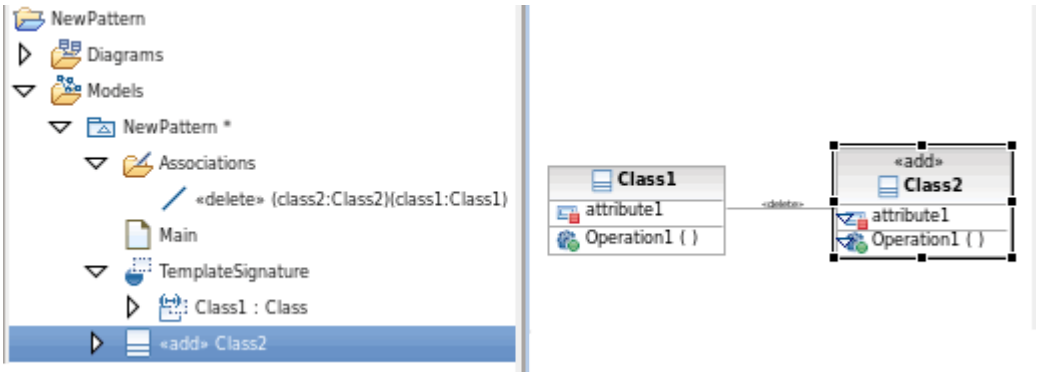

Ilustración 26. Ejemplo de elementos con estereotipos del Perfil PatternSpec aplicados

3.2.2 Estereotipos disponibles

La siguiente tabla muestra los estereotipos definidos en el perfil y la transformación que especifican:

Estereotipo	Especificación
new	El elemento será creado en el modelo de destino sin importar si éste ya existe o no en el modelo fuente.
add	El elemento se añadirá al modelo de destino si éste no existe ya en el modelo fuente.
delete	Si el elemento existe en el modelo fuente, éste será eliminado.
optional	El elemento parametrizado no tiene porqué ser sustituido por un elemento del modelo fuente, cuando es así no se crea en el modelo final.
rename	Un elemento del modelo fuente que substituye a otro en el patrón tomará el nombre de este último.

3.2.3 Limitaciones de los estereotipos

A pesar de que en el diseño del perfil RSA permite aplicar los estereotipos de *PatternSpec* a cualquier elemento, existen limitaciones a los elementos que pueden ser usados en el diseño de los patrones. Para poder usar elementos distintos es necesario ampliar la funcionalidad de la herramienta a través del código.

Las restricciones de cada perfil a los elementos que pueden extenderlos en el diseño de los patrones se muestran en la siguiente tabla:

Estereotipo	Elementos
new	*Class, Interface, Package, Association, Generalization, InterfaceRealization, Attribute (en las clases parametrizadas de las plantillas), Operation (en las clases e interfaces parametrizadas de las plantillas).*
add	*Attribute (en las clases parametrizadas de las plantillas), Operation (en las clases e interfaces parametrizadas de las plantillas).*
delete	*Association, Attribute (en las clases parametrizadas de las plantillas), Operation (en las clases e interfaces parametrizadas de las plantillas).*
optional	*ParameterableElement (elementos parametrizados de las plantillas)*
rename	*ParameterableElement (elementos parametrizados de las plantillas)*

3.3 Renombrar elementos mediante parámetros

En el diseño de los patrones se pueden añadir unas notaciones específicas para indicar que elementos del patrón que serán creados en el modelo final, adoptarán el nombre de elementos del modelo fuente como parámetro en sus propios nombres. La forma de indicar que algún elemento toma el nombre de otro para el suyo propio es insertando una cadena de caracteres que indicará que en su lugar se insertará el nombre de un elemento del modelo fuente.

La cadena tiene la forma ***$reference_element$.*** Los símbolos **$** indican que se inserta un parámetro. ***Reference_element*** es el elemento del patrón que sirve de referencia para tomar el nombre que se inserta en el nombre del elemento a renombrar. El elemento de referencia tiene que ser un elemento parametrizado del patrón; y el nombre se toma del elemento del modelo que juega el rol de dicho elemento.

Ejemplo de caso de uso:

Este caso sería el de un elemento que es creado en el modelo porque está así especificado en el patrón y queremos que el nombre del elemento creado tenga parcial o totalmente el nombre de un elemento de nuestro modelo.

La figura inferior muestra un ejemplo de patrón con un parámetro de plantilla y su correspondiente elemento parametrizado “Interface” y otra clase “$Interface$_wrapper” que será creada y generalizará a la clase que asuma el rol de Interface.

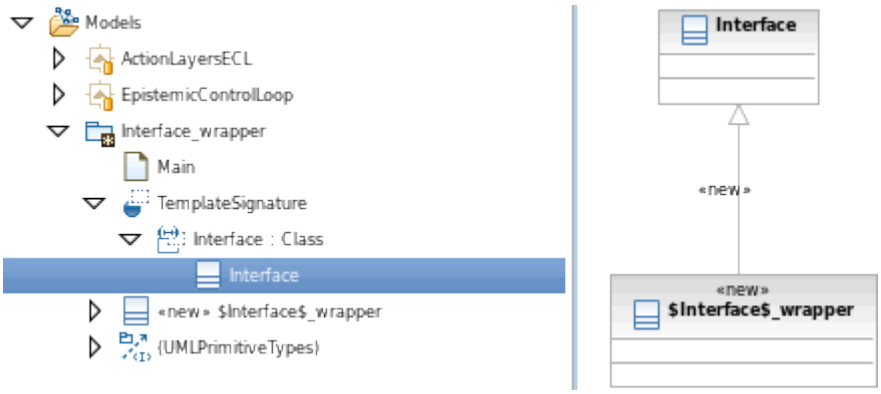

Ilustración 27. Ejemplo de patrón con renombre de elementos a través de parámetros

La subcadena “*_wrapper*” permanecerá intacta (también se podía haber añadido una subcadena al principio), mientras que la parte entre los símbolos “$” (incluidos) será sustituido por el nombre que pasamos como parámetro. La referencia que se da para el nombre es el elemento *Interface*. Este elemento, al ser un elemento parametrizado, aportará como parámetro el nombre del elemento real del modelo que asume el rol de dicho elemento. Es decir, el nombre que se añade a la clase creada será el de la clase que asume el rol de *Interface*. (En el caso de que haya más de un elemento real, se tomará el del primero de ellos).

Aplicando el patrón a un modelo formado por una clase *Class1* que asume el rol de *Interface* el resultado tras la transformación sería el siguiente:

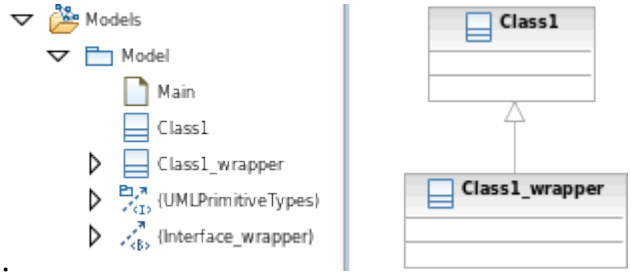

Ilustración 28. Resultado de aplicar patrón con renombre de elementos a través de parámetros

El resultado es que la clase *Class1* asume el rol de *Interface*, y se crea una clase que hereda de ella. Esta clase, toma el nombre de *Class1* para insertarlo en el suyo propio resultando *Class1_wrapper*.

4. Ampliación de la funcionalidad

En los siguientes apartados se explicarán algunos puntos de extensión de la funcionalidad de la herramienta, así como algunas reglas para el correcto funcionamiento de las extensiones que se realicen dentro del código.

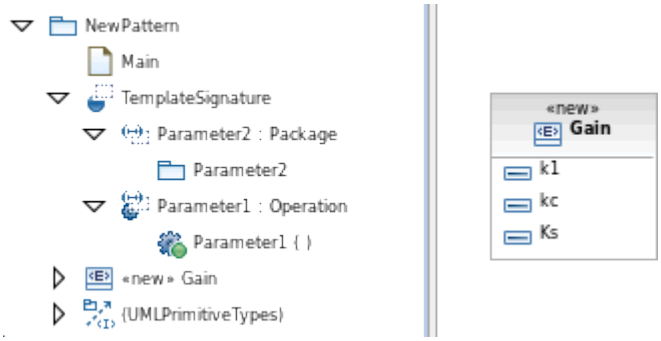

Ilustración 29. Ejemplo de extensión de parámetros de plantilla y elementos estereotipables

La idea es que se puedan realizar pequeñas extensiones en los elementos que pueden ser parametrizados o estereotipados sin tener que conocer en profundidad la estructura del código. En el patrón de la ilustración 29 hay dos elementos de plantilla *Parameter1 (Operation)* y *Parameter2 (Package)*, que en caso de ser substituidos, sólo aportarían un elemento que juega un rol; no serían transformados de modo alguno pues no está contemplado en el código de PatternApplier. También hay un elemento *Gain (Enumeration)* que está estereotipado para ser creado en el modelo sobre el que se aplica el patrón *(New)*. Este tipo de elemento no está soportado por la herramienta, por lo que sería necesario realizar una extensión.

Son casos simples de extensiones que pueden surgir durante el diseño de un patrón. Aún así, se considera que el usuario de este punto posee conocimientos de cierto nivel sobre las librerías que permiten la generación de UML mediante programación y el Eclipse Modeling Framework para manejar los elementos de los modelos UML.

4.1 Extender elementos parametrizables

Como se ha mencionado en el punto 3.1.2, los elementos que se pueden usar como parámetros de plantilla están limitados. Si se quiere poder usar en el diseño alguno distinto a los indicados será necesario añadir la semántica de la transformación que produce ese tipo de elemento en un patrón. Esto se hará en el código de PatternApplier.

Para ello existe la clase *PatternParameters*. Esta clase maneja las substituciones realizadas en el modelo sobre el que se aplican los patrones. Cuando la herramienta es ejecutada, tras reconocer el modelo y los patrones que interactúan, la primera acción de transformación que realiza es resolver los parámetros de plantilla.

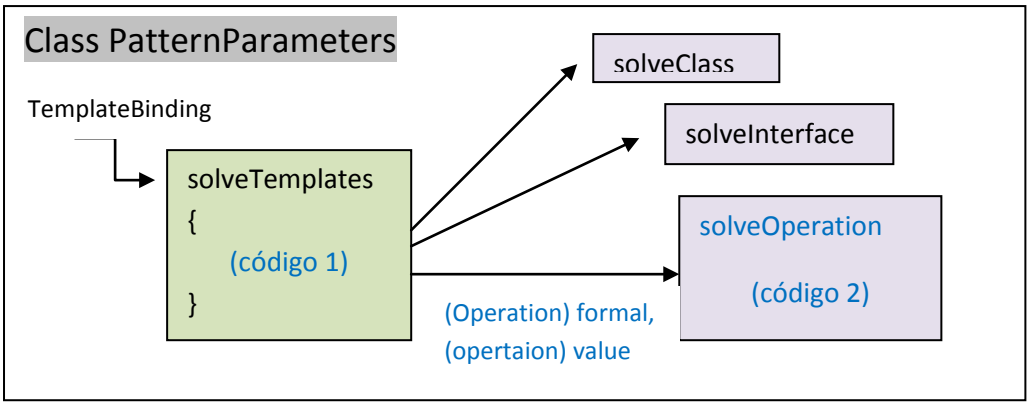

Ilustración 30. Diagrama de funcionamiento de la clase *PatternParameters*

El primer método de la clase es *solveTemplates*. Este método recorre las substituciones realizadas interpretando el tipo de elemento de la plantilla. Si se quiere algún tipo nuevo se tendrá que usar el punto de extensión al final del método (Ilustración 30). Por ejemplo, si tenemos un tipo de elemento *Operation*, la extensión sería:

```
else if (value instanceof Operation) {

      solveOperation((Operation) formal, (Operation) value);

}
```

Código 1. Extensión de tipos de parámentros de plantilla

Hay que indicar el tipo de elemento del que es instancia el elemento de la plantilla, en este caso *Operation*. Así, cuando se dé el caso, los elementos implicados serán tratados con el método especialmente diseñado para operaciones *solveOperation*. A este método se le pasan el elemento *formal*, que es el elemento parametrizado del patrón, y *value*, que es el elemento del modelo que lo substituye. Es importante hacer "casting" del tipo de elemento de la plantilla en el paso de los parámetros *(Operation) formal*, *(Operation) value*.

Después habrá que implementar el método *solveOperation*, que se añadirá en la misma clase *PatternParameters*. El usuario tendrá que completar el método en base a la semántica que se quiera dar a este nuevo tipo en el diseño de patrones. Se requerirá conocimiento del paquete org.eclipse.uml2.uml para manejar los elementos UML del modelo y del patrón. A continuación se muestra la definición del método.

```
protected void solveOperation(Operation formal,Operation actual)
{
      * Código del método
}
```

Código 2. Forma del método solveOperation

Los métodos ya implementados para otros tipos de elementos pueden servir de orientación. Esta es la estructuración básica para los elementos de plantilla añadidos en el diseño del patrón. En los puntos 4.3 y 4.4 se explicarán otras clases de la herramienta para dar soporte durante la transformación de los elementos del modelo.

4.2 Extender elementos estereotipados

Como pasaba con los parámetros de plantilla, los elementos que pueden ser estereotipados durante el diseño de los patrones están limitados también. Por ejemplo, elementos que pueden ser marcados con el estereotipo *new* para que sean creados en el modelo de destino. Será necesario pues ampliar también el código para añadir nuevos elementos a los indicados en el punto 3.2.3.

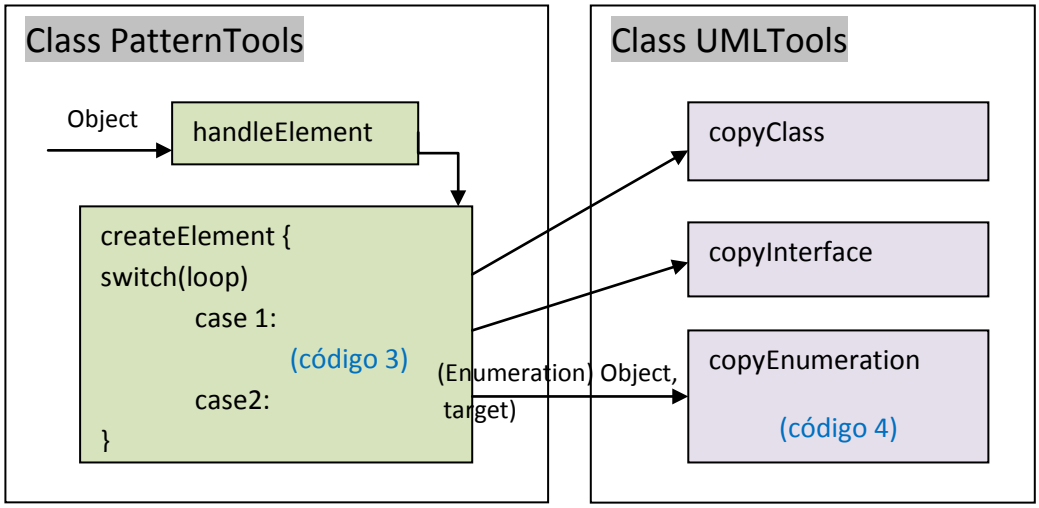

Ilustración 31. Diagrama de creación de elementos nuevos en el modelo de destino

Estas modificaciones tendrán comienzo en la clase *PatternTools*. Tras realizar las tareas oportunas con los parámetros de plantilla, la aplicación pasa a trabajar con los elementos estereotipados del patrón. La herramienta analizará dos veces los elementos estereotipados y realizará las transformaciones necesarias. Se hará dos veces porque se hará distinción entre dos tipos de elementos según sean elementos individuales (clases, interfaces, paquetes,…) o sean elementos de tipo asociación. Esta distinción se realiza ya que en ocasiones, para poder crear elementos del segundo grupo, será necesario haber creado ciertos elementos del primero. Por ejemplo, para crear una generalización de una clase C1 a otra clase C2 que se crea en el modelo porque está así indicado en el patrón, será necesario que se cree antes la superclase C2, para así poder crear la asociación de generalización de C1 a C2.

En el método *createElement* habrá que hacer una extensión para dar soporte a un nuevo tipo de elemento. Suponiendo que queramos poder añadir un tipo de elemento como *Enumeration*, habría que añadir un caso en el primer bucle *loop* (ilustración 31), pues se consideraría un elemento individual.

```
if (object instanceof Enumeration) {
      UMLTools.copyEnumeration((Enumeration) object, target);
      break;
}
```

Código 3. Extensión de tipos de elementos que se pueden crear en el modelo de destino

Con esta extensión se podría crear un elemento *Enumeration* en el modelo sobre el que se aplica el patrón. Si se quisiera que la funcionalidad fuese borrar en lugar de crear, habría que habría que implementar un método para chequear la existencia del tipo (punto 3.4) y en tal caso eliminarlo. Se puede tomar como ejemplo para ello, el caso de *Association* que permite eliminar asociaciones.

En el código superior se llama a un método perteneciente a otra clase de la herramienta, *UMLTools*, que se tendrá que implementar también. Esta clase que proporciona métodos para copiar elementos UML se explicará en el punto siguiente. Cabe destacar, que al método se le han de pasar como parámetros el elemento del patrón que se toma como referencia para copiar *(Enumeration) Object*, y el modelo *(target)* que donde se realiza la copia del primero. La forma en que se crean elementos en el modelo dependerá siempre de el propio elemento que queremos crear por lo que será necesario tener los conocimientos necesarios para ello.

4.3 Clase UMLTools

Esta clase de la herramienta está implementada como una librería estática que proporciona métodos para copiar elementos UML. Los métodos están implementados de forma que al pasarles como parámetros un elemento y un paquete o modelo UML se crea una copia del primer elemento es este paquete o modelo. Se considerará copia hasta el nivel que el usuario quiera que compartan unas mismas características.

En el ejemplo propuesto en el punto anterior, había que implementar un método llamado *copyEnumeration* dentro de la clase *UMLTools.* La forma que debería tener es la siguiente:

```
static Enumeration copyEnumeration(Enumeration enum,
                                   Package target ) {
      newenum = target.createOwnedEnumeration(null);
                  .
                  .
                  .
      * Más código del método
}
```

Código 4. Forma del método copyEnumeration en la clase UMLTools

El usuario tendrá que completar el método copiando las características que crea oportunas del elemento *enum* al elemento *newenum.* Si fuese necesario, podrían usarse otros métodos de *UMLTools* para completar el método; este sería el caso de que el elemento que se copia contenga otros tipos de elementos UML.

4.4 Clase CheckTools

Esta clase proporciona métodos para realizar comprobaciones o localizar elementos dentro del modelo sobre el que se le aplica el patrón.

Uno de los métodos que puede ser necesario a la hora de ampliar la funcionalidad de la herramienta es el método *searchNamedElement.* Este método localiza en el modelo sobre el que se aplican los patrones un elemento a partir de otro elemento o de un nombre. Por ejemplo, si necesitamos buscar las clases que relaciona en el modelo una asociación, tomaremos las clases del patrón que están relacionadas con la asociación en el patrón y usando *searchNamedElement* se asignará la clase equivalente para el modelo. Esta clase equivalente puede presentar 3 casos: primero, que la clase haya sido copiada del patrón al modelo y entonces existiría una con el mismo nombre en el modelo; segundo, que la clase que se toma como referencia para la búsqueda sea un elemento parametrizado, por lo que el método buscaría en el modelo sobre el que se aplica el patrón el elemento que juega el rol del elemento parametrizado y ese sería el que asignaría; y finalmente, si no se da ninguno de casos anteriores, se devolvería como resultado de la búsqueda el propio elemento leído en el patrón. Esto significaría que es un elemento perteneciente a otro modelo.

En esta clase se pueden crear también métodos para comprobar si ciertos elementos existen dentro del modelo sobre el que se aplica el patrón. Esta igualdad está sujeta siempre a las características que se consideren de interés. Es el caso de los métodos *searchAssociation, searchOperation, y searchProperty.* En el caso de *searchAssociation,* la asociación se buscará dentro un modelo dado. En los otros dos métodos, el elemento se buscará en una lista de elementos del mismo tipo. La forma de búsqueda estará sujeta a la finalidad de uso del método.

Siguiendo con el ejemplo del elemento *Enumeration,* se presenta la forma que tendría el método *searchEnumeration.* Se puede tener un elemento de tipo *Enumeration* en el patrón y querer buscar un elemento del mismo tipo en el modelo, que se considere igual hasta cierto nivel de detalle. Será necesario pasar como parámetros al método, el elemento *Enumeration* del patrón cuya copia se está buscará en el modelo, así como el modelo o el paquete del modelo donde se realizará la búsqueda. Se analizarán todos los elementos del modelo; y si alguno cumple todas las condiciones para considerarlo igual al elemento de referencia *Enumeration,* se devolverá este elemento encontrado. Si tras analizar todos los elementos del modelo no se ha encontrado ningún elemento que satisfaga todas las condiciones, se devolverá *null*.

```
static Enumeration searchEnumeration(Enumeration enum,
                                     Package target) {

      for (Iterator<EObject> iter = target.eAllContents();
            iter.hasNext();) {


            EObject object = iter.next();
            if (* Comprobaciones de igualdad) {
                        .
                        .
                        .
                  return testenumeration;
            }
      }
      return null;
}
```

Código 5. Forma del método searchEnumeration en la clase CheckTools